Lab Manual – 04

Course Title: Algorithm Lab
Course Code: CSE 223
Semester: Spring 2020
Topic: Coin change and Knapsack
(Dynamic and Greedy approach)

::::Course Teacher::::
Masud Rabbani
Lecturer
Dept. of CSE, DIU

::::Prefect::::

Muhaiminul Islam
Student of DIU
Id: 163-15-8473
Mail: muhaiminul15-8473@diu.edu.bd

Muhiyminul Islam
Student of DIU
Id: 172-15-10187
Mail: muhiyminul15-8473@diu.edu.bd

Hasan Imam Bijoy
Student of DIU
Id: 182-15-11743
Mail: hasan15-11743@diu.edu.bd

Md Mahbubur Rahman
Student of DIU
Id: 182-15-11742
Mail: mahbubur15-11742@diu.edu.bd

Dynamic Programming

Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of sub-problems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial. For example, if we write simple recursive solution for Fibonacci Numbers, we get exponential time complexity and if we optimize it by storing solutions of sub-problems, time complexity reduces to linear.

Greedy Approach

A **Greedy algorithm** is any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum. In many problems, a greedy strategy does not usually produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a globally optimal solution in a reasonable amount of time.

Coin Change

Introduction: Basically, coin changing is nothing but solving a problem which is related to coins, where there is a target number and given some amount of coins and we have to determine the optimal solution to reach the target. And it can be done by using dynamic programming and greedy approach also.

Applications:

- Suppose you are a shopkeeper and there is customer in your shop. And you have to return total 11 taka to him. And you have coins of 1 5,6, and 8. Now suppose you are willing to give it by using minimum number of coins . So that's where it will come handy

- And moreover, Coin change problem is actually a very good example to illustrate the difference between greedy strategy and dynamic programming.

Complexity:

- Dynamic Programming: O(k*n)
- Greedy: O(nlogn)

Outcome:

The **coin changing problem** addresses the question of finding the minimum number of coins (of certain denominations) that add up to a given amount of money. It is a special case of the integer knapsack problem, and has applications wider than just currency. So, after implementing it we can have a optimal solution for our desired problem.

Advantages and disadvantages (Dynamic Programming):

Advantages are –

- For the various problems in area such as inventory, chemical engineering design, and control theory, Dynamic Programming is the only technique used to solve the problem.
- It is well suited for multi-stage or multi-point or sequential decision process.
- It is suitable for linear or non-linear problems, discrete or continuous variables, and deterministic problems.

Disadvantages are:

- No general formation of Dynamic Program is available; every problem has to be solving in its own way.
- Dividing problem in sub problem and storing inter mediate results consumes memory.

Advantages and disadvantages (Greedy Method):

Advantages are –

- They are easier to implement.
- They require fewer computing resources.
- They are much faster to execute.
- Greedy algorithms are used to solve optimization problem.

Disadvantages are:

- Their only disadvantage is that they do not always reach upto optimum solution.
- On the other hand, even when the global optimum solution is not reached, most of the times the reached sub optimal solution is very good solution.

Knapsack Problem:

Introduction: The **knapsack problem** or **rucksack problem** is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. The name "knapsack problem" dates back to the early works of mathematician Tobias Dantzig (1884–1956).

Applications:

- The branch and bound **algorithm** to solve the **0-1 knapsack problem**, one of the most widely-used combinatorial optimization algorithms, is used to capture the customer values and the discrete characteristics of loads. The objective of the model is to maximize customer values within given supply capacity.
- (Real life application) What is our thinking just a night before exams? Get more marks by less study in the remaining amount of hours. Right? So, here we are trying to optimize our efforts for getting more marks. Now, we analyze our previous question papers and decide that which chapters have more value and which chapters have less. We also have a sense that how much time it will take to complete the chapter by checking the number of pages of the chapter in the book. Here we are trying to maximize the marks by selecting the chapters whose questions have a high probability of asking in the exams. And also we have to consider the hours required to complete these chapters. Now we check all the combinations of chapters and their values to find the list of chapters that needs to be studied and also, add the hours required to study these chapters so the added hours should not exceed the hours we have for the exam.

Complexity:

- O(nW) where, n is the number of items and W is the capacity of **knapsack**

Outcome:

Commonly **used** example **problem** in combinatorial optimization, where there is a need for an optimal object or finite solution where an exhaustive search is not possible. So by using 0/1 knapsack technique we can find optimal solution for our desired problem.

Advantages and disadvantages (Dynamic Programming):

Advantages are –

- It is used to find all of the existing solution if there exist for any problem.

Disadvantages are:

- Multiple function calls are expensive.
- Requires large amount of space as the each function state needs to be stored in the memory.
- Cannot take portion of an object.

Fractional Knapsack Problem:

Applications:

Real life based: One early application of knapsack algorithms was in the construction and scoring of tests in which the test-takers have a choice as to which questions they answer. For small examples, it is a fairly simple process to provide the test-takers with such a choice. For example, if an exam contains 12 questions each worth 10 points, the test-taker need only answer 10 questions to achieve a maximum possible score of 100 points. However, on tests with a heterogeneous distribution of point values, it is more difficult to provide choices. Feuerman and Weiss proposed a system in which students are given a heterogeneous test with a total of 125 possible points. The students are asked to answer all of the questions to the best of their abilities. Of the possible subsets of problems whose total point values add up to 100, a knapsack algorithm would determine which subset gives each student the highest possible score.

Complexity:

- Fractional Knapsack has **time complexity** O(NlogN) where N is the number of items.

Outcome:

Since in this technique we have rights to take some portion or fraction of the objects so there will be less wastages than 0/1 knapsack as it's rule was to take the entire object or none.

Advantages and disadvantages :

Advantages are –

- Can take fraction or portion of an object so it derives to optimal solution.

Disadvantages are:

- Can be solved by only greedy approach.
- Implementation is difficult.

Basic Code of Coin Changing(Dynamic Approach)

```
#include <stdio.h>
#define MIN(x, y) (((x) < (y)) ? (x) : (y))
const int INF = 100000;
int coin_change(int d[], int n, int k)
{
   int M[n+1];
   M[0] = 0;

   int i, j;
   for(j=1; j<=n; j++)
   {
      int minimum = INF;
      for(i=1; i<=k; i++)
      {
         if(j >= d[i])
         {
            minimum = MIN(minimum, 1+M[j-d[i]]);
         }
      }
      M[j] = minimum;
   }
   return M[n];
}
int main()
{
   int d[] = {0, 1, 2, 3};
   printf("Coin Need: %d\n", coin_change(d, 5, 3));
   return 0;
}
```

Practice Problem:

1. Suppose you want to pay back someone with 16 taka and you have unlimited supply of 1 taka, 2 taka, 8 taka and 12 taka notes. Your target is to use the minimum number of notes to payback the said amount of money. That means how many notes you chose, write the program.

Input	Output
Coin [] = 1,2,8,12} Change : 16	Coin Need : 2

2. Suppose you are given the coins 1 taka, 5 taka, and 10 taka with Total Taka = 8 taka, what are the total number of permutations of the coins you can arrange to obtain 8 taka.

Input	Output
Total Taka : 16 Coins [] = {1,5,10}	Coin Need : 3

Basic Code of Fractional Knapsack(Dynamic Approach)

```
#include<stdio.h>
int max(int a, int b)
{
   return (a > b)? a : b;
}
int knapSack(int W, int wt[], int val[], int n)
{
   int i, w;
   int K[n+1][W+1];

   for (i = 0; i <= n; i++)
   {
      for (w = 0; w <= W; w++)
      {
         if (i==0 || w==0)
            K[i][w] = 0;
         else if (wt[i-1] <= w)
            K[i][w] = max(val[i-1] + K[i-1][w-
wt[i-1]],  K[i-1][w]);
         else
            K[i][w] = K[i-1][w];
      }
   }

   return K[n][W];
}
```

```
int main()
{
   int i, n, val[20], wt[20], W;

   printf("Enter number of items:");
   scanf("%d", &n);

   printf("Enter value and weight of items:\n");
   for(i = 0; i < n; ++i)
   {
      scanf("%d%d", &val[i], &wt[i]);
   }
   printf("Enter size of knapsack:");
   scanf("%d", &W);

   printf("%d", knapSack(W, wt, val, n));
   return 0;
}
```

Practice Problem:

1. Suppose, you go to a market with a knapsack, which maximum capacity is 5kg. There four items with weight and value. Then find the maximum profit and print it.

Item	Weight	Value
Gold	2	12
Diamond	1	10
Platinum	3	20
Rubi	2	15

Input	Output
Total Items: 4 Weight [] = {2,1,3,2} Value [] = {12,10,20,15} Knapsack Weight: 5	Maximum Profit: 37

2. You are given some items with weight and value, put these items in a knapsack of capacity 50 to get the maximum total value in the knapsack. Note that we have only one quantity of each item. Follow the above approach and print the maximum profit.

Item	Weight	Value
1	100	20
2	50	10
3	150	30

Input	Output
Total item :3 Weight [] = {100,50,150} Value [] = { 20,10,30}	Maximum Profit: 250

Basic Code of Coin Changing(Greedy Approach)

```
#include <stdio.h>
void insertion(int A[], int x)
{
   int item, i;
   for(i = 1; i < x; i++)
   {
      item = A[i];
      int j = i - 1;
      while(j >= 0 && A[j] < item)
      {
         A[j+1] = A[j];
         j = j - 1;
      }
      A[j+1] = item;
   }
}
```

```
int main ()
{
   int m,i,back, total_coin;
   int taka[100];
   printf("Enter total coin : ");
   scanf("%d", &total_coin);
   printf("Enter coin : ");
   for(i = 0; i < total_coin; i++)
   {
      scanf("%d", &taka[i]);
   }
   insertion(taka,total_coin);
   printf("Enter taka : ");
   scanf("%d", &back);
   printf("\nMinimum coin : \n");
   for(i=0; i<total_coin; i++)
   {
      if(taka[i]<= back)
      {
         m=back/taka[i];
         printf("%d coin %d times\n",taka[i],m);
         back=back%taka[i];
      }
   }
}
```

Practice Problem:

1. Suppose you went to a market, then you gave 15 tk note to the sales person, and he returned some of the currency (1,7,7,10S) that he had in abundance. Then write the code which is find optimal solution.

Input	Output
Coin [] = {1,7,7,10} Change Amount: 15	Coin Need: 10 coin 1 times 1 coin 5 times

2. Suppose, you have coin changing machine. Which has huge amount of coin. Now if any one gave some money then it is return some of coins. Follow the above approach and find the which coin could be needed and how many times?

Input	Output
Enter Number of Coin: 5 Coin [] = {2,5,3,4,6} Change Amount: 12	Coin Need: 6 coin 2 times

Basic Code of Fractional Knapsack(Greedy Approach)

```
# include<stdio.h>
void knapsack(int n, float weight[], float profit[], float capacity)
{
   float x[20], tp = 0;
   int i, j, u;
   u = capacity;
   for (i = 0; i < n; i++)
      x[i] = 0.0;
   for (i = 0; i < n; i++)
   {
      if (weight[i] > u)
         break;
      else
      {
         x[i] = 1.0;
         tp = tp + profit[i];
         u = u - weight[i];
      }
   }
   if (i < n)
      x[i] = u / weight[i];
   tp = tp + (x[i] * profit[i]);
   printf("\nThe result vector is:- ");
   for (i = 0; i < n; i++)
      printf("%f\t", x[i]);
   printf("\nMaximum profit is:- %f", tp);
}
int main()
{
   float weight[20], profit[20], capacity;
   int num, i, j;
   for (i = 0; i < num; i++)
   {
      scanf("%f %f", &weight[i], &profit[i]);
   }
   printf("\nEnter the capacityacity of knapsack:- ");
   scanf("%f", &capacity);
   for (i = 0; i < num; i++)
   {
      ratio[i] = profit[i] / weight[i];
   }
   for (i = 0; i < num; i++)
   {
      for (j = i + 1; j < num; j++)
      {
         if (ratio[i] < ratio[j])
         {
            temp = ratio[j];
            ratio[j] = ratio[i];
            ratio[i] = temp;
            temp = weight[j];
            weight[j] = weight[i];
            weight[i] = temp;
            temp = profit[j];
            profit[j] = profit[i];
            profit[i] = temp;
         }
      }
   }
   knapsack(num, weight, profit, capacity);
   return(0);
```

float ratio[20], temp; *printf("\nEnter the no. of objects:- ");* *scanf("%d", &num);* *printf("\nEnter the wts and profits of each object:- ");*	*}*

1. Suppose you enters a house for robbing it. You can carry a maximum weight of 60 kg into your bag. There are 5 items in the house with the following weights and values. Then find the maximum profit if you can even take the fraction of any item with you? Write the program.

Item	Weight	Value
01	5	30
02	10	40
03	15	45
04	22	77
05	25	90

Input	**Output**
Total Items: 5 Weight [] = {5,10,15,22,25} Value [] = {30,40,45,77,90} Knapsack Weight: 60	Maximum Profit: 230 tk

Basic Code of Fibonacci Number(Dynamic Approach)

```
#include<stdio.h>
int fib(int n)
{
   int f[n+2];
   int i;
   f[0] = 0;
   f[1] = 1;
   for (i = 2; i <= n; i++)
   {
      f[i] = f[i-1] + f[i-2];
   }
   return f[n];
}
int main ()
{
   int n;
   printf("Enter Any Number : ");
   scanf("%d",&n);
   printf("Fibonacci Number : %d", fib(n));
   getchar();
   return 0;
}
```

Practice Problem:

1. Write a program to display nth term of Fibonacci series of n terms.

Input	Output
Enter Any Number: 9	Fibonacci Number: 34
Enter Any Number: 13	Fibonacci Number: 233

2. Are you know the Fibonacci series? If yes, given a positive value with t. Then find the solution & also print it.

Input	Output
Test Case: 3 Number 1: 1 Number 2: 2 Number 3: 5	Result: Fibonacci :1 Fibonacci :1 Fibonacci :5

Edited By

Md Atiqur Rahman

"Every people have a specialty, work on it, you will be a hero on your life"

'Happy Coding'

☺